3

RADC-TR-88-133, Vol I (of three)
Final Technical Report
June 1988

THE C² SYSTEM INTERNET EXPERIMENT

BBN Laboratories Incorporated

Kenneth J. Schroder, James C. Berets, Ronald A. Mucci and Richard E. Schantz

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss AFB, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-88-133, Vol I (of three) has been reviewed and is approved for publication.

APPROVED:

THOMAS F. LAWRENCE
Project Engineer

APPROVED: Raymond P. Urtz Jr

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: John A. Ritz

JOHN A. RITZ
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTD) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notice on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD A200 167

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS
N/A

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE
N/A

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release; distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
6251

5. MONITORING ORGANIZATION REPORT NUMBER(S)
RADC-TR-88-133, Vol I (of three)

6a. NAME OF PERFORMING ORGANIZATION
BBN Laboratories Incorporated

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION
Rome Air Development Center (COTD)

6c. ADDRESS *(City, State, and ZIP Code)*
10 Moulton Street
Cambridge MA 02238

7b. ADDRESS *(City, State, and ZIP Code)*
Griffiss AFB NY 13441-5700

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
Rome Air Development Center

8b. OFFICE SYMBOL *(If applicable)*
COTD

9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F30602-84-C-0140

8c. ADDRESS *(City, State, and ZIP Code)*
Griffiss AFB NY 13441-5700

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.
62702F	5581	21	70

11. TITLE *(Include Security Classification)*
THE C^2 SYSTEM INTERNET EXPERIMENT

12. PERSONAL AUTHOR(S)
Kenneth J. Schroder, James C. Berets, Ronald A. Mucci and Richard E. Schantz

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM Aug 84 TO Mar 86

14. DATE OF REPORT *(Year, Month, Day)*
June 1988

15. PAGE COUNT
44

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
12	07	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Distributed Operating System, Interoperability, Heterogeneous Distributed System, System Monitoring and Control, Survivable Application

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This is the Final Technical Report representing work performed for the C2 System Internet Experiment project. The C2 Internet Experiment project is a complementary effort to the CRONUS Distributed Operating System (DOS) project. Under the C2 Internet Experiment, we designed and built a prototype distributed application to demonstrate CRONUS concepts and to provide an environment for evaluating CRONUS' current suitability for supporting distributed applications, especially applications that pertain to the command and control environment. Both the C2 Internet Experiment and CRONUS DOS projects are part of a larger program being supported by the Rome Air Development Center (RADC) to design, develop and support an appropriate technology base for building the types of distributed systems which are expected to be fundamental in future command and control applications and elsewhere.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Thomas R. Lawrence

22b. TELEPHONE *(Include Area Code)*
(315) 330-2158

22c. OFFICE SYMBOL
RADC (COTD)

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Table of Contents

1. INTRODUCTION

This is the Final Technical Report representing work performed for the C^2 System Internet Experiment project. The C^2 Internet Experiment project is a complementary effort to the Cronus Distributed Operating System (DOS) project. Under the C^2 Internet Experiment we designed and built a prototype distributed application to demonstrate Cronus concepts and to provide an environment for evaluating Cronus' current suitability for supporting distributed applications, especially applications that pertain to the command and control environment. Both the C^2 Internet Experiment and Cronus DOS projects are part of a larger program being supported by the Rome Air Development Center (RADC) to design, develop and support an appropriate technology base for building the types of distributed systems which are expected to be fundamental in future command and control applications and elsewhere.

The Cronus system development project has been ongoing since 1981[1]. The goal of this effort is to develop an architecture, an operating system, and software development tools to support large scale distributed systems and applications. The development to date has produced a Cronus testbed which serves to provide application developers with exposure to Cronus concepts and software for developing their distributed applications and to provide an opportunity for direct experience with the effects of distribution on their applications. This testbed is comprised of operational computer systems and software located at BBN in Cambridge, Mass; the testbed is currently being extended to include additional resources located at RADC in Rome, NY. The Cronus testbed currently supports the C^2 Internet development and operation. Work on the testbed development and on enhancements and extensions to the Cronus system are being performed concurrently with the development of the C^2 Internet Experiment application.

The C^2 Internet effort has focused on developing a representative application chosen from the command and control area. The particular application area chosen is a set of battle management functions whose integration is representative of larger systems, and whose function can be enhanced through the use of distributed system architectures and technologies. Application development to date has proceeded in a top-down fashion, emphasizing the interfaces and functions of components that comprise the application and their use in a distributed environment by other components. Once this architectural framework is established, the fidelity and functionality of the individual components can be enhanced. We believe that this approach reflects the actual need to be able to integrate components whose functions and capabilities will evolve after their interface and functions have been established initially.

[1] See Cronus Functional Definition [Schantz85] for further details.

1.1. Project Objectives

There are a variety of uses of the Cronus system which have been explored in the experimental application. One important role of Cronus is to support and control interoperability between otherwise independent functions and resources available on a heterogeneous collection of host systems. Support for this interoperability is also meant to facilitate the evolution from existing but largely non-integrated computing resources towards an effective integration and management of resources and services which can capitalize on information and control shared through interconnection. A second role is to facilitate the development of a multi-host implementation of particular functions or subsystems as an alternative to single host implementations. Such distributed implementations can provide special properties, such as scalability and survivability, which are not possible through single host implementations. A third role is to facilitate the long-term evolution of newly constructed, large-scale distributed systems. This includes the replacement of existing components with new, more technologically advanced counterparts, and the integration of new, possibly unanticipated functions and resources.

The C^2 Internet Experiment was also meant to serve as a vehicle for demonstrating the potential role of distributed system technology in supporting C^2 applications. In this role, the development of the C^2 Internet Experiment application served as a basis for evaluating the support provided by the Cronus distributed operating system and its tools. It also served as a preliminary basis for evaluating the relevance of distributed system techniques to the C^2 Internet environment. To be an effective basis for this evaluation, it was essential that the particular application domain we chose be visibly relevant to the C^2 Internet environment and offer many opportunities to incorporate and to present distributed system benefits. Because the level of effort allocated to the effort was limited, it was important that the application be adequately demonstratable without excessive attention to details of particular application components. The application also needed to be extensible to serve as a foundation for future enhancements through technological improvements and additional application development effort.

Beyond the relevance of the application to the C^2 environment, we hope that the application-independent nature of much of the underlying system support is also clearly recognizable. Distributed systems offer an opportunity both to support the interoperation of an existing heterogeneous collection of resources and to allow functional specialization of subsystems for which particular technologies can be exploited, including concurrent multiprocessors, symbolic processors and high-performance display processors and workstations. Distributed systems also offer many benefits to the support of particular system components and their use of the available system resources. These include:

- Survivability: the entire system and essential functional components continue to be available when hosts and other physical system components are unavailable; when multiple failures occur, performance should degrade gradually (*graceful degradation*), not suddenly.
- Scalability: system capacity can be increased by adding additional physical resources but without substantial change to the software or system architecture.

- Resource Management: resources are assigned to clients, and system components are placed, to ensure efficient use of these resources and components; configuration choices can be changed to improve performance and ensure efficient use of all available resources.
- Evolvability: hosts, resources, subsystems and other components of the system can be changed to accommodate redesign and technological improvements without requiring substantial changes to other components of the system.
- Extensibility: new components can be added to the system without requiring substantial changes to other system components beyond what is required to support the use of the new component.

The C^2 Internet Experiment also provides an opportunity to test and evaluate the relevance and effectiveness of the Cronus approach, and to test and evaluate system and application support within the context of complex, prototype C^2 applications with requirements similar to projected uses. Our evaluation has focused on the qualitative aspects, such as the suitability of the Cronus object model and development methodology to system development; and the adequacy and flexibility of the underlying system and development tools. Some instrumentation and quantitative evaluation has also been performed, largely oriented toward establishing a small set of general purpose tools for identifying areas for more detailed analysis.

The process of evaluation also serves to establish a source of feedback, gained from experience in matching actual application requirements to Cronus support mechanisms and from developing the prototype application, to help guide the direction of future Cronus system enhancement, extension and repair. This has been especially useful to developers, who provide support for various system attributes such as survivability, global resource management, and multi-cluster operation, as well as providing a means for the hands-on evaluation of an initial implementation of these properties within the context of the application. This also guided the effort to further formalize, extend and refine the methodology used for designing distributed applications, and the tools and system support used to implement distributed applications.

1.2. Project Summary

Because of the application's central role, one of the first tasks accomplished under this effort was the design of an appropriate application, summarized in a paper by Berets, et. al. [Berets85] and described in more detail in the C^2 Internet Functional Definition [Berets86]. The design included both identification and specification of the functional components needed to support the application, and a set of prototype scenarios of use within this application domain for demonstration and evaluation purposes. This application definition was then submitted for review and accepted.

Using the Cronus development tools and other Cronus support mechanisms, we refined the component designs and implemented initial versions of the application functions, as described in the System/Subsystem Specification [Berets86a]. Initially, development focused on a subset of key functions, both to achieve an early demonstration capability, and to reinforce the utility of an evolutionary approach to system development. The collection of components we have developed has substantially increased, representing a preliminary implementation simulating the full functionality appropriate to the application, emphasizing those aspects which focus on distributed system technology. To date, our development has successfully accommodated changes in the application implementation, both revisions to existing components and the addition of new components, without dramatic impact on other system components. These functions will continue to evolve over time.

A working configuration of C^2 Internet Experiment application has been operational for several months, and is used regularly in demonstrations. We intend to continue development of the application during ongoing and future efforts, both to enhance the properties of existing components and to introduce new functions and system architectures into the application environment. One major extension is the recent integration of a second testbed cluster into the application environment; communication between the two clusters is provided by the DARPA Internet, including the Arpanet and Wideband Satellite networks. Other changes will include modifying existing components to introduce survivability and resource management in areas where they were not essential for the initial demonstrations, adding symbolic processors to support experimentation with artificial intelligence based approaches to application functions, and possibly adding more extensive use of distributed database technology. Changes to the underlying Cronus system environment will include better user interface support, enhancing the development tools and possibly introducing better debugging support.

The C^2 Internet Experiment has served to guide our emphasis in the ongoing Cronus development work. In particular, our emphasis on relatively high-level tool support has grown, with less emphasis on extending the capabilities of lower level communications primitives. In the course of developing the application components, and in using the Cronus system and application development tools, there have been numerous enhancements and refinements to the Cronus system and tools; a number of other areas, such as database and program development support, have been identified for future development. The underlying Cronus system and tools have evolved substantially during the course of this effort, and we have found that the system and application has quite readily afforded the necessary changes. We believe that the integration of the system and application development activities, in which the C^2 Internet Experiment provides a basis for evaluation and goal setting for the underlying Cronus activities, has been quite effective.

1.3. Report Overview

The rest of this report is divided into three sections. Section 2 discusses the overall application design and development process. This includes a description of the design methodology, clarified with examples from the C^2 Internet Experiment application, and a description of the use of Cronus system support and distributed software development tools in relation to development methodology. It briefly reviews the Functional Definition [Berets86], System/Subsystem Specification [Berets86a], and demonstration application [Berets86b] produced as part of the development effort and identifies how distributed system benefits are visible in the demonstration.

Section 3 discusses our experimental findings. This includes both strong points and weaknesses. Where weaknesses have been corrected, we include a discussion of what was discovered and how it was corrected. Particular emphasis is given to the existing Cronus software, including the development tools, and some discussion of the status of the experiment configuration and demonstration capabilities.

Section 4 discusses our conclusions and recommendations. In general, the experiment has been extremely successful, based both on our subjective evaluation of the current Cronus system in supporting the development of distributed applications, and in the response we have received from people who have seen the software demonstrated. The Cronus system model and the Cronus system software and development tools were readily applied to the application architecture; the application software itself was developed by a combination of experienced and novice project staff. The application provided the complexity needed to exercise the Cronus DOS testbed, demonstrates integrated functioning of resources on a variety of hosts and makes extensive use of existing Cronus utilities. The areas where weaknesses were discovered were not surprising, reflecting areas where little design and development effort had been focussed. Problems which might prevent effective use of the system were fixed immediately, concurrent with continued application development. Other enhancements which would make development more comfortable or convenient have been scheduled for later development.

A final section includes references to other C^2 Internet and Cronus reports and publications.

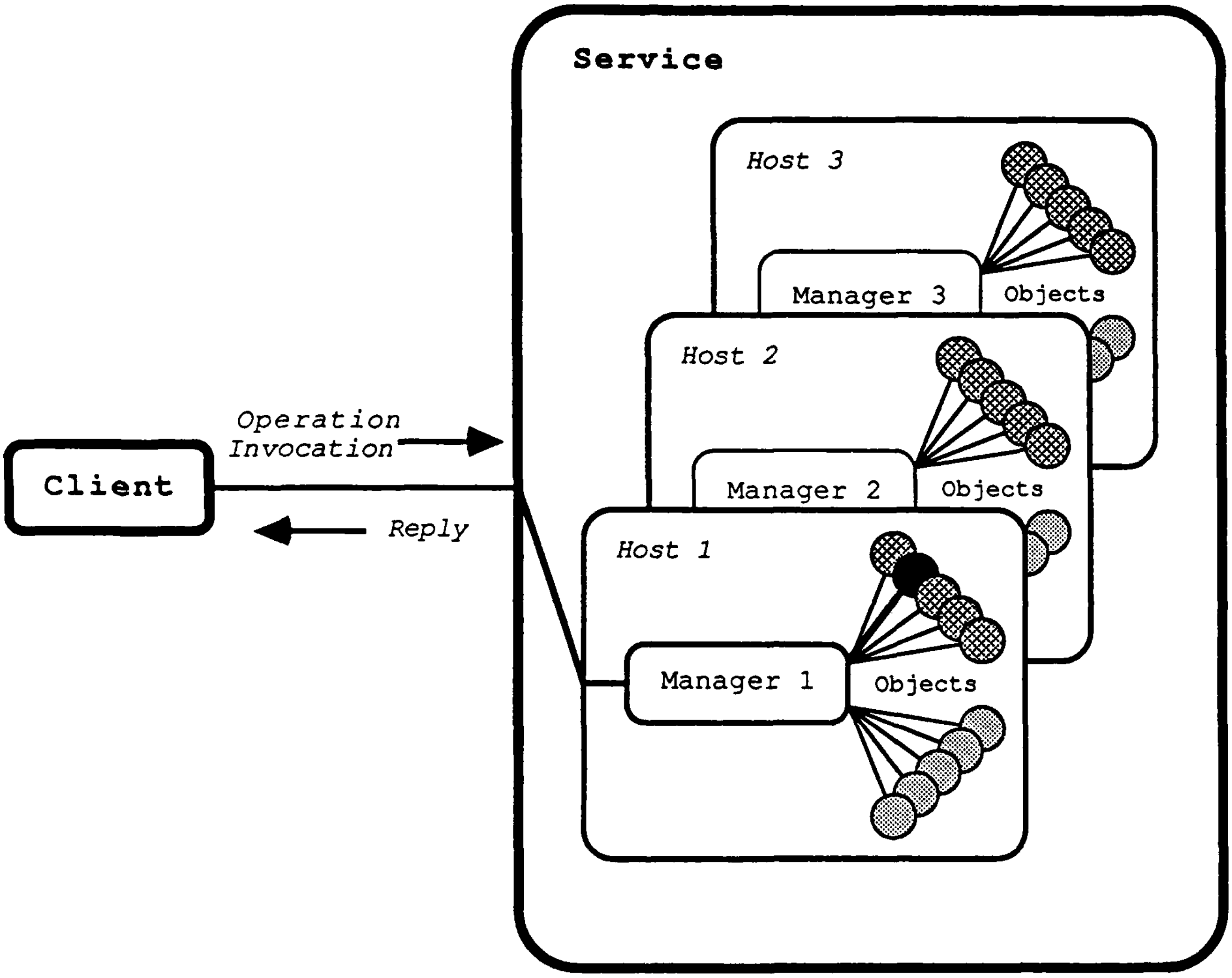
2. DISTRIBUTED SYSTEM DEVELOPMENT METHODOLOGY

The basic goal of the Cronus distributed system development methodology is to simplify and facilitate the process of developing distributed systems and applications. The methodology defines a series of loosely coupled stages, ranging from functional decomposition and architectural design in the early stages through implementation and evolutionary revision in the later stages. The tasks of considering operational functions, ensuring various component properties such as survivability and scalability, choice of implementation language, selection of support hardware, and later revision are decoupled in this methodology. The resulting series of stages are collectively less complex and easier to manage than a less structured, ad-hoc approach.

The choice of object orientation as the architectural model of this methodology serves to modularize the system and to decouple the development of independent system components. In the Cronus model, applications are conceived as a collection of *services* managing various kinds of resources, and *clients* which submit requests to use or affect the resources provided by the services. Each service is responsible for a set of *object types*; each object type describes a kind of resource (*objects* of the given type) and an appropriate set of *operations* available to clients for affecting objects of that type. The specification of the essential, unchanging properties of the interface to a service and the discrete identification of resources as named objects independent of their location, promote application architectures which quite naturally support interoperability and portability, which allow existing application components to be applied to new, similar problems, and which afford smoother evolution of an existing application as new functional needs are identified. Object types can also be used to describe the characteristics of a generic kind of resource, such as a file. *Subtypes* will *inherit* the characteristics of such a *generic* parent type. The subtype specification may also identify additional object characteristics and operations.

Services are implemented by one or more *managers* which are responsible for carrying out operations on objects. Each manager exists as a process on one of the host systems and is responsible for a subset of the objects provided by the service. Managers may also act as clients of other services. Managers for a particular service may cooperate to provide resource management and survivability properties for the objects provided by the service. These strategies often include maintaining duplicates of an object at several sites to provide alternative paths, both in the interest of improved efficiency for resource management, and in the interest of ensuring survivable access if one copy of an object should become unavailable.

The diagram in Figure 1 shows how a collection of managers support the operations on objects provided by a particular service. In this case, a client program has invoked an operation on an object supported by one of the managers. The route to the desired object is determined and selected by the Cronus kernel and the operation is routed to the appropriate manager on the destination host. That manager then carries out the required operation processing and returns a reply to inform the client of the results.



Cronus Object Model
Figure 1

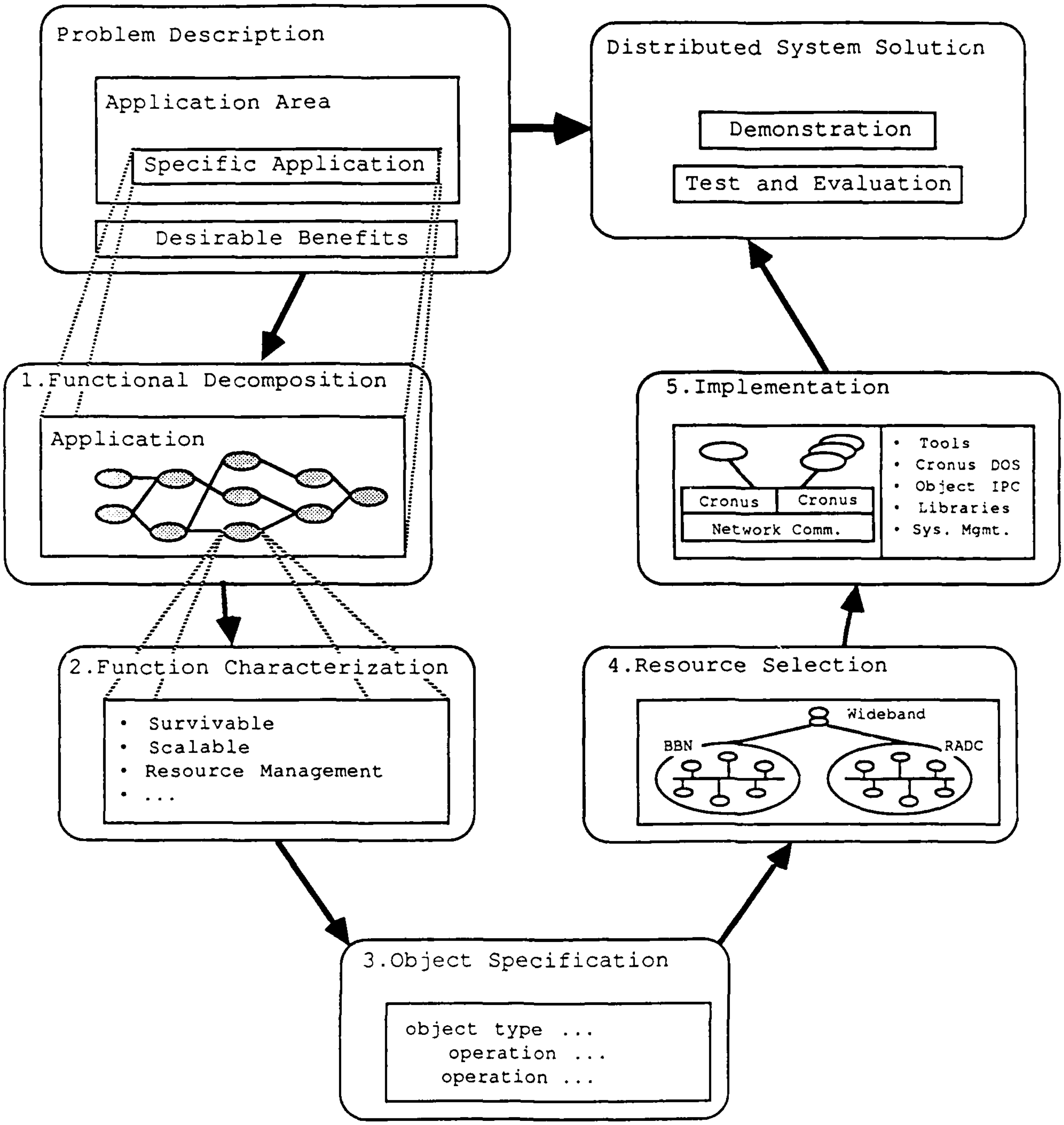
Tools are also provided to support the software development of client and manager programs. These tools include specification tools that facilitate work by the program developer writing software in selected programming languages. Cronus system services and support libraries provide resources and mechanisms as a foundation for the implementation and operation of a distributed application. Guidelines are given for deciding how support should be provided for properties such as survivability, scalability and resource management. The tools and libraries often provide default mechanisms for supporting these properties. Additional details of these tools can be found in [Gurwitz86, Sands86].

The object oriented methodology used to develop the C^2 Internet Experiment application was originally devised to guide the development of the Cronus Distributed Operating System (DOS). A discussion of how the object model was applied to the Cronus DOS Architectuie can be found in [Schantz86]. The model proved quite effective in organizing the types of resources commonly found in traditional operating systems, including processes, directories, files, users and groups; and for extending support for these resources through a distributed collection of heterogeneous host systems, providing controlled remote resource access, survivable operation, scalability, and resource management. The model also proved effective for supporting a collection of tools for program development and system management. Our goal in applying this methodology to the C^2 Internet Experiment was to discover whether the techniques could be effectively applied in an application context, particularly in the C^2 domain.

We currently divide the development process into a series of five phases:

1. Divide Application by Functional Decomposition
2. Identify Distributed Function Characteristics
3. Specify Components in Terms of Cronus Object Model
4. Map Functional Elements onto Appropriate Host Resources
5. Implement and Debug

These steps are diagramed in Figure 2. The first two phases of the process involve decomposition of the problem into distinguishable functional elements and specification of particular properties that each element must support. Then, in phase three, the resources associated with and operations performed by each functional element are specified as Cronus object types, and their relationship to Cronus support for properties such as survivability and scalability is established. In the fourth phase, particular physical resources are identified to support the operation of the function, with managers being appropriately placed to support use of the associated objects. During the last phase, the functions are implemented, either by using existing Cronus system components in new ways, by developing new type managers, or by using a combination of both system components and new type managers. Integration with existing application components follows, and continuing evolution of the components through further changes can occur. For the initial development of an application or of an application component, these phases are conducted sequentially. However, once an application is in place, its continuing evolution will often involve repeating the process for particular components and for collections of components. As a result, the development of most installed applications cannot be characterized as being



C^2 Internet Experiment Development Methodology
Figure 2

completed, but rather as evolving with each component at an independent stage of development.

These phases are described in more detail in the following sections. Each section briefly describes the characteristics of the development stage and includes examples taken from the C^2 Internet Experiment application.

2.1. Divide Application by Functional Decomposition

The functional decomposition follows the application area specification. Primarily, this identifies the major subsystem elements, defines their role in the overall processing tasks of the application, and describes how the elements interact and rely upon each other when performing the tasks required of the application. This is a fairly high-level specification. Each identified element should be functionally autonomous, representing a portion of the system that can be sensibly isolated from the other components, and which might be suitable for functionally specialized computer support. However, since Cronus mechanisms operate transparently across host boundaries, with local and remote communication handled uniformly, and because software which does not rely on specialized computer support can be written to be relatively portable, we can essentially ignore computer related issues at this stage.

A collection of typical application scenarios are identified during the functional definition phase. Reviewing the expected behavior of the system in these scenarios helps to ensure that all critical elements are identified during the design phase, when changes can be more easily accommodated. This will also help determine the order in which components should be built, with more commonly used components typically developed before those whose use is more limited.

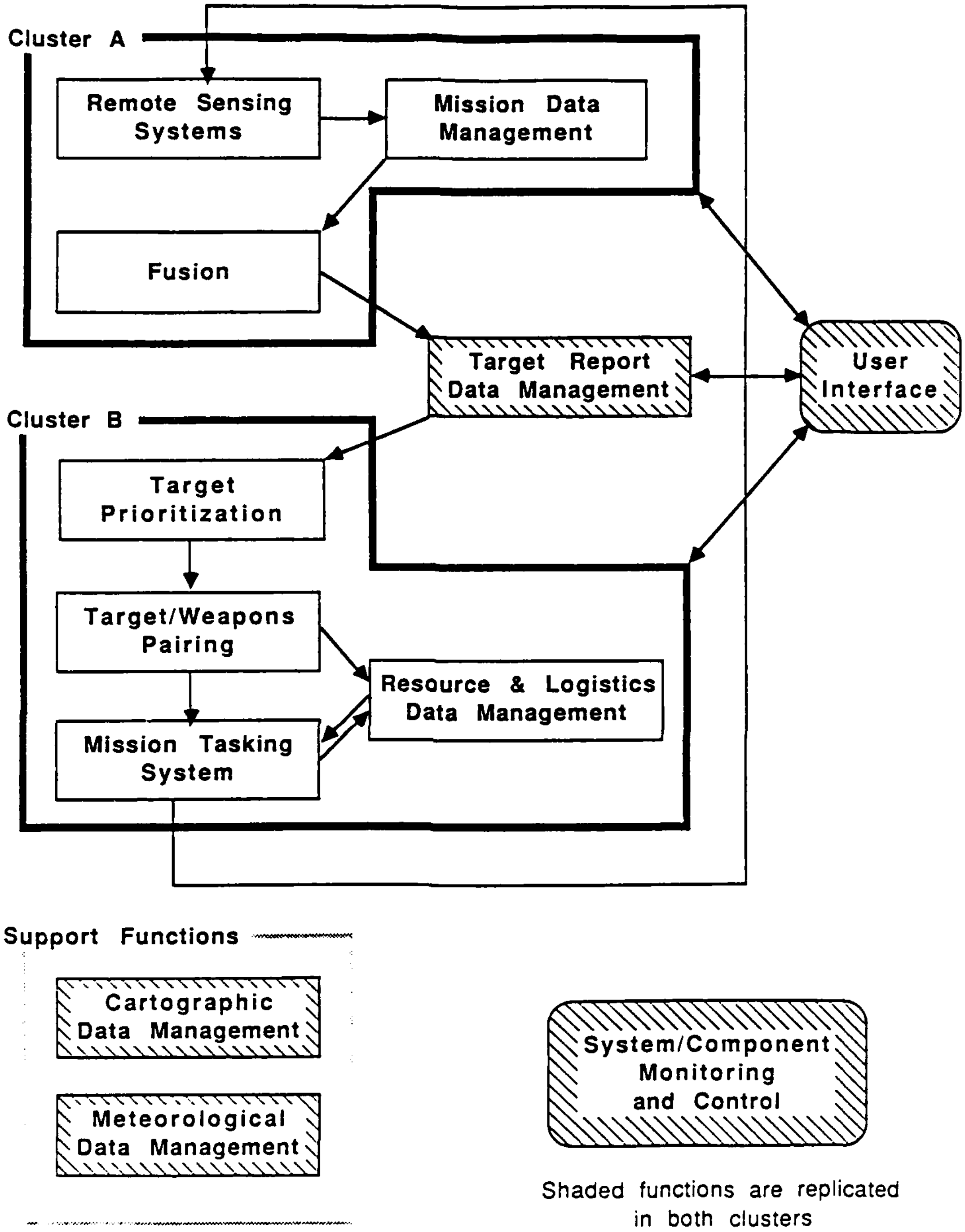
The functional elements of the C^2 Internet Experiment are displayed in Figure 3. The application consists of gathering and reviewing situation information provided by mobile sensing systems to track the movement of incoming targets. The application also includes some of the support functions needed to plan and initiate an appropriate response. Three, essentially different, kinds of processing are cyclically performed.

Reconnaissance where image and other data describing the field situation is sensed, and transmitted for storage and analysis;

Analysis where targets are detected and identified in preparation for planning appropriate actions;

Mission Planning where the field situation is displayed for review, responses to the targets are evaluated, and appropriate actions are initiated.

The process repeats both to determine the effectiveness of whatever action was initiated and to detect new threats.


Cluster A
Remote Sensing Systems
Mission Data Management
Fusion
Target Report Data Management
User Interface
Cluster B
Target Prioritization
Target/Weapons Pairing
Resource & Logistics Data Management
Mission Tasking System
Support Functions
Cartographic Data Management
Meteorological Data Management
System/Component Monitoring and Control
Shaded functions are replicated in both clusters

C^2 Internet Experiment Processing Functions
Figure 3

Support functions were also identified, initially to include cartographic and meteorological data management. These data resources are used in a variety of ways for the other components, and operate independently of the principal processing flow. System and application monitoring and control is also required to track the status of the distributed system components.

The processing was further decomposed into finer grained activities. The particular functions we chose were not meant to be comprehensive, because we had limited resources to build the system and had limited knowledge of the application domain. Our goal was to elaborate the specification in enough detail to guide implementation, and to cover enough of the salient features of the application area to support a credible demonstration. The more realistic the application description, the greater the relevance of the benefits achieved will be to the observer of the demonstration. However, since it is only a demonstration, realism had to be weighed against the cost associated with building the necessary components. We also believe that additional functions can be added to the application, in an evolutionary manner, after the essential processing functions have been installed.

Reconnaissance is decomposed into *Remote Sensing Systems* and *Mission Data Management.* This separation captures the fact that a variety of sensing systems provide data to one logically central mission data management system. The Mission Data Management System is logically central both to the sensors which store the data, and to the fusion and other processing functions which retrieve the data for analysis.

Analysis is comprised of three components, distinguished by function and use: Fusion Processing, Target Report Data Management, and Target Prioritization. *Fusion Processing* correlates target detection information from several sensor missions to yield improved estimates of where targets are located, to determine what their trajectories are, and to identify the kind of vehicle the target represents. The resulting target reports are stored for later processing by the *Target Report Data Management System.* This information is then reviewed by an operator using the *Target Prioritization* system, which assigns priorities reflecting how quickly a target should be considered and dealt with.

Mission Planning is a much more interactive processing activity, initially divided into three components. During *Target/Weapons Pairing,* particular weapons are assigned to intercept selected targets. *Resource and Logistics Data Management* is used to determine available weapons and reconnaissance resources, and to track their use. The *Mission Tasking System* is used to dispatch selected weapons and sensors on their missions.

2.2. Identify Distributed Function Characteristics

The initial functional description serves to decompose the processing tasks to be performed by the system. Our next phase is to elaborate the component descriptions to identify the distributed system properties each component should possess. Once these characteristics are specified, a combination of standard Cronus mechanisms and customized support mechanisms will be selected to provide these characteristics as part of the implementation phases. These properties.currently include:

- Access Control
- Scalability
- Survivability
- Resource Management

One goal of the Cronus system is to support these properties through mechanisms which are convenient to the developer's use. These mechanisms are often provided through the tools and are selected through a high-level specification language. The underlying support mechanisms may also be used by the developer when particular features are not available through the higher-level interfaces. Both approaches were used in developing the C^2 Internet Experiment application.

For an example of how these properties apply to a system resource, consider a collection of files which store the source code for a project. The owner of each file should be able to restrict access to his files and should be able to modify the restrictions periodically. As the project grows, additional storage space must be added, scaling the available amount of space for file storage. Also, we might like to ensure that the files are always available, possibly keeping duplicate (*replicated*) copies of the files so that if one host fails another can supply the data. And finally, when the files are spread across a variety of hosts, both as the scale increases and as replicated file copies are introduced, automated management of the file space is needed. This includes chosing the original file placement, matching clients to particular file copies, and relocating files to improve performance as the number of files and their use changes.

We consider how these special properties apply to each of the C^2 Internet Experiment application components. To review, the components are:

- Remote Sensing Systems
- Mission Data Management
- Fusion (Target Data Correlation)
- Target Report Data Management
- Target Prioritization
- Target/Weapons Pairing

- Resource and Logistics Data Management
- Mission Tasking System
- Cartographic Data Management
- Meteorological Data Management
- User Interface
- System/Component Monitoring and Control

Access control, scalability, survivability and resource management are appropriate properties for each of these components. However, because our development resources and testbed size were limited, we chose to focus on particular considerations for each component, chosing the properties we believed were most important and, for each service, chosing resources where the Cronus system support for these properties could be most readily evaluated and presented in a demonstration. Access control of operations on all application resources is supported through the standard Cronus mechanisms. For example, access control is used to restrict who may schedule and reschedule the missions for particular sensor systems. The standard access control mechanisms were tailored to restrict access to the Resource and Logistics Data Management, which catalogs available weapons systems and other tactical resources. In this case, the standard mechanisms where used to control access to information about particular weapons systems and a tailored mechanism was developed to ensure that queries on the database would not reveal the existence of or information about weapons systems which the client did not have the right to examine. Scalability was important to the Remote Sensing Systems since they represent physical components in the field that may be added in indefinite numbers. This makes scalability and resource management essential to the Mission Data Management System, which collects and records partially processed data from each sensor system. Scalability is less of an issue in other components because they need to process only a subset of the total data at any one time, and because they are not required to record or process the data in real-time, but can rely instead on the data recorded by the Mission Data Management System. Survivability is essential for the Mission Data Management data recording functions because a remote sensor may become unavailable during the course of its mission. However, if several Mission Data Managers are configured, the data retrieval functions of a particular Mission Data Manager is less important, as long as the failure of one manager affects the availability of only a small portion of the data for a particular region or mission.

There were a number of considerations for the User Interface and for the simulation control subset of the Experiment Monitoring and Control. The user interface was built so that several could be running simultaneously, either to monitor different regions of a single simulation or to monitor various regions of different simulations. For the simulation control, all components must be survivable and the commands that control a simulation must be available through several access points. We planned to use the monitoring and control facilities offered by the Cronus system to support monitoring and control of the application. Because this was an experimental context, we were less concerned with access control to these monitoring and control functions during this stage of development than we would be in an operational system; we do allow for their later addition.

After reviewing our needs for the User Interface and simulation control we determined that two additional components would be needed: a simulation clock to control the progress of experiment simulations; and a target simulator to provide simulated target trajectories. The simulation clock provides a synchronized time of day and multiple simulation clocks whose rate and time setting can be varied under operation control. The target simulator stores target trajectories entered by an operator and allows clients to request a list of the target that pass through a particular geographic region during a selected time interval. Both of these components can be used to orchestrate multiple, concurrent C^2 Internet Experiment simulations, and can be applied to other simulation applications in the future.

2.3. Specify Components in Terms of Cronus Object Model

Having identified and characterized the functional components of the application, the next phase is to identify services that will support the functions identified in the previous phase, and to specify the distributed system properties that each of these functions must support. Similar types should be identified since they are candidates for introducing a type hierarchy. Characteristics common to several types can be specified by a generic parent type and particular subtypes can then elaborate and tailor particular aspects of the parent types. This use of a type hierarchy ensures compatible interfaces and allows clients using those types to inte operate more freely.

In the Cronus operating system, services include processes, files, directories, authentication, and many others. Files and directories are actually generic types representing more than one service: files represent both Cronus migratable files and COS files; directories represent both the Cronus catalog and COS directories. In the C^2 Internet Experiment, most services correspond to functional components such as Remote Sensor Systems and Mission Data Management. Many of these application services rely on support from a system service, such as the catalog or file system. Operation of the application itself is performed by the interaction of these application and system services. In larger applications, functional components would likely be supported by several interacting services.

Perhaps the major activity at this stage is specifying the *operations* that may be used to affect particular objects, and the *instance variables* which characterize each individual object. For an example from the C^2 Internet Experiment consider the simulation clock service. Managers of this service support two major types of objects: a collection of independent simulation clocks called *timers*; and a synchronized time of day clock. The simulation clocks may be created and removed. Each timer is distinguished by the time when it was last restarted and the rate at which it is running; from this information, the current simulation time value can be calculated. Each timer may also have a stop time—the timer will be automatically stopped if the specified time occurs. Operations on timer objects may be used to set its rate and stop time, and to start and stop it. There is only one time of day clock, which is replicated by all simulation clock managers. One of these managers is selected, by election among the managers, to be responsible for keeping all the other managers synchronized. Operations on the time of day clock include changing the time of day and manually selecting the master clock.

Election and synchronization operations are also supported and restricted to use by the simulation clock managers.

The Cronus tools offer a specification language for formally identifying thes‹ object instance variables and operations. It also supports the definition of *canonical types*. Canonical types are used to communicate information in a host independent format, unlike object types which are used to identify system and application resources. Thus, an operation may take as parameters several canonical types such as an the new time or rate setting for a simulation clock. Descriptions of the system object types can be found in the Cronus User Reference Manual [Sands86]. The object specifications for the C^2 Internet Experiment application components are included in the System/Subsystem Specification [Berets86a]. These descriptions are machine translated from the actual object type specifications to improve their readability for use as reference material.

The Resource and Logistics Data Management System is readily described as operations on a collection of object instances. Currently, two object types are used to specified the resources managed by this service: vehicles (jeeps, tanks, planes, etc.) and airfields; others will be added in the future. The use of the resource and logistics data has some additional requirements beyond those of the sensor systems. We still require operations on individual vehicle and airfield object instances, but we must also support query operations that scan all available objects and return a list of the appropriate subset of objects. This is used, for example, when an operator needs to know which fighters are within flying range of a particular area. These queries are supported through operations on the *generic object*, which is used in this case to denote a collection of objects: the generic vehicle object is used to denote the collection of all known vehicles; the generic airfield object is used to denote the collection of all known airfields[2].

The choice of objects is less obvious in the case of the Mission Data Management service where operations may refer to different collections of data. The Mission Data Management service provides data storage and query based retrieval to clients, essentially a small database facility. It is essentially a special purpose distributed database service which might be replaced by a more general purpose database service when one becomes available. Typically, a deployed sensor system will submit a list of target relevant detections, identifying the estimated target position at the time of detection. The most natural object boundary for this time series data is either individual detections, contemporaneous detections, or the detections associated with a particular mission. The processing and analysis components will typically request a list of all detections that

[2] Use of the generic object as a reference to all objects of a particular type is not yet fully supported by the underlying system. By default, the generic object will refer to the objects handled by the manager which initially receives the request. To affect the comprehensive set of objects managed by a service requires either additional support by the managers to propagate the request, or requires the client to contact all managers through a broadcast mechanism. The major use of the generic object in this way has been in searching for the object identifier associated with a symbolically named object, as is done by the catalog manager mapping pathnames to identifiers and by the authentication manager mapping principal and group names to principal and group objects. Special support is implemented in these managers: the authentication managers maintain copies of all objects at each site for survivability, so it does not matter which manager processes the request; the catalog manager forwards requests to appropriate managers as the pathname is decoded. Because the Resource and Logistics manager was not a major focus of the effort, we chose to expedite implementation by operating just a single instance of the manager, so no additional support for cooperation among the managers was required. In the future this function will be facilitated by additional support from the Cronus tools and mechanisms.

occurred in a particular region over some period of time. The most natural object boundary for this time series data is either individual detections or detections within the specified region that occurred contemporaneously or during the specified time period. To match these boundaries seems to require that all detections be stored separately. However, storing data as a large collection of small objects might allow objects to be scattered at several different sites and would complicate the mechanics of performing query requests because a large number of sites must cooperate and each must search a relatively large number of objects. We chose to exploit the fact that mission schedules tend to focus on particular regions and so retrieval requests are likely to select information that corresponds to a set of missions. This allows us to store the object information as missions and to perform our retrievals by extracting detections from appropriate missions. The results of a retrieval identify the list of detections with each detection marked to indicate which mission supplied the data.

This approach to the Mission Data Management requires additional peer cooperation among mission data managers so that the data provided is comprehensive. The specification of this *peer protocol* is also part of this stage of development. In particular, when a query request is recei.ed by one manager for the service, that manager broadcasts the query request to its peers so that all managers collect the relevant information from their mission data objects. The peer responses are then sent to the controlling manager which combines the peer data with its own response and returns the comprehensive report to the client. Peer cooperation is also used to support a *round-robin* scheduling algorithm to select which manager will be responsible for recording data for a new mission. When a manager receives a request to store data for a new mission, it broadcasts a request to its peers to determine which one has least recently stored a mission. The manager responding to the initial invocation then forwards the invocation to the manager chosen by the resource management strategy. This resource management algorithm is also used to reassign the recording site when a manager fails[3].

2.4. Map Functional Elements onto Appropriate Host Resources

Once the functions have been specified, we identify special host characteristics that their operation may require. Since Cronus supports a wide variety of heterogeneous host systems, currently including Digital Equipment Vax Systems, SUN Workstations, and more recently a Symbolics Lisp Machine, one type of system may offer substantial advantages over others. Even within similar types of systems, one site may offer more disk space or an array processor or other specialized resource.

For the C^2 Internet Experiment, most elements can be freely located, at the operator's discretion. This is largely because the host independent Cronus system mechanisms promote component portability, because host boundaries are made transparent through object based communication mechanisms and canonical data representations, and because intra-host and inter-host communication mechanisms are provided through a

[3] In the current implementation the data which have already been recorded are lost but can be recovered from raw data kept by the sensors. The mission data objects could be replicated for survivability, but we felt that experimenting with a variety of recovery techniques better served the goals of the experiment.

uniform interface. Generally, components which were actively being developed were located on the workstations, which had the most productive development environment, and components which were more mature were migrated to less productive development environments, such as the timesharing systems and C70 minicomputers. This included migrating components from 16/32-bit workstation architectures to 20/40-bit BBN C/70 architectures with little or no modifications.

We are also in the process of establishing an additional cluster at RADC which will share resources with the existing cluster at BBN. These clusters are interconnected by the DARPA Internet, including a high bandwidth connection through the Wideband Satellite Network. This will allow the RADC cluster to make remote use of symbolic processors and other resources unique to BBN's installation, in addition to providing additional support for functions operating within the RADC cluster. This will also allow us to experiment with functional specialization at the cluster level, perhaps focusing one cluster on reconnaissance and preliminary analysis functions and the other on forecasting and operational and review functions

The one major exception to portability is the user interface and MCS console interface, which run on workstations. These components make extensive use of the graphics support facilities offered on the workstations since more portable user interface foundations have not been developed for Cronus so far. Also, in the future, more sophisticated fusion processing might be provided by the Lisp Symbolic Processors and various data management functions might be provided by a set of specialized database machines.

2.5. Implement and Debug

There are two important phases to our program development phase: initially, key components were developed and tested in isolation; next, this initial subset of application components were integrated. At this stage, development became incremental, adding new functions to enhance the existing components and adding new components to the configuration. The initial configuration consisted of the components needed to support the simulation: a target simulator and a replicated simulation clock to establish a global time base for synchronizing the events in the simulation. Then additional components that relied only on these components where added: the sensor manager and the resource and logistics data manager. Once initial versions of these components were completed, the remaining functions were added as needed and were enhanced as we gained experience with using the system.

Development of the components relied both on the application development tools and on the services provided by the Cronus DOS. Cronus applications are organized around a high-level object type specification language and a companion manager specification language. These are described in detail in Section 4 of the Cronus User Reference Manual [Sands86]. The object type specification language accepts a source language that describes the characteristics of each object instance, the operations that can be invoked on objects of the given type, and the parameters to the object operations. The object type descriptions are stored by a type definition service: another program takes the stored object type description and generates code for invoking operations by clients and for handling operation dispatching, access control and other customary manager operation processing actions.

The system support services that we used included authentication and access control, the replicated catalog, constituent operating system file and directory access, and system and application configuration management. Appropriate status reporting operations are supported by all managers to allow the Cronus monitoring and control system to be used for both system and application resources.

3. EXPERIMENTAL FINDINGS

The C^2 Internet Experiment set out with two primary goals:

- evaluating the current Cronus implementation for its suitability in building an exemplary distributed application, and
- providing a demonstration capability for Cronus through an application in the command and control area.

The application we developed works and demonstrates many advantages of the distributed system architecture in the context of C^2 applications. Many areas of Cronus were evaluated by building the C^2 Internet Experiment application. The Cronus system served effectively in supporting this particular application and in facilitating development. We also believe that the methodology can be effectively applied to distributed applications in general, and the Cronus mechanisms will support a wide variety of applications outside the C^2 Internet environment as well.

The evaluation also proved useful in revealing some of the deficiencies of the Cronus environment. Where deficiencies were discovered, our goal was to test whether the Cronus facilities could be changed to suit the needs of the application developer within the framework of the existing system and system model. In general, our conclusions have been positive, calling largely for elaboration and refinement of the existing system.

This section will discuss our findings.

3.1. Methodology

One of the significant benefits Cronus provides is its methodology, supported by Cronus system services, tools and mechanisms. The Cronus design philosophy provides a coherent abstraction under which a wide variety of distributed application attributes can be united. This uniformity extends to cover both the application and underlying Cronus system components, allowing application components to easily make use of available system components and promoting an economy of mechanisms which the system must support, which lead to simpler and smaller systems. The Cronus object model abstraction naturally supports the decomposition of a problem into a number of communicating components. This in turn supports a multi-person development environment in which both software and hardware evolution is easily accommodated and can track different and evolving support technology bases. This is demonstrated in the C^2 Internet application development.

The application development tools were especially valuable, providing a tangible link from the abstract decomposition and design steps performed early in the process to the configuration and implementation steps performed later in the process. High level specification reduced the amount of effort both through its support for generating client/manager interface code and common manager structure functions, and through an established format for specifying the objects and operations supported by the application. The common format allowed managers to be designed before beginning implementation and allowed type interfaces to be read and understood much more easily.

3.2. Available System Facilities

Cronus provides a variety of system facilities to application developers. Many of the ones used during the C^2 Internet experiment were discussed earlier in this report. These included system supported services such as the catalog manager, tool supported mechanisms such as access control and survivability through object replication, and support by other system services and libraries. These facilities were adequate and effective to support the application development.

Within the context of the application, some of the available Cronus mechanisms were found to be inadequate. These included both bugs, such as failure of tools support replication, and areas where an existing mechanism needed to be more elaborate. Bugs were generally minor and were corrected. Where mechanisms needed improvement, the C^2 Internet Experiment needs were evaluated against other development requirements and the improvements were made as soon as could be accommodated. We include here two examples of the kinds of improvement that were required.

The Catalog Manager originally used a locking scheme that only allowed updates to a replicated directory when all copies of the directory were available. Thus, if a host supporting the catalog manager failed or was shut down for servicing, it was likely that updates to several replicated directories would be refused. Allowing this limitation expedited the development of the original replicated catalog implementation. After suggestions by various users of this facility were made, the catalog was modified to use a *version vector* technique allowing updates to be made to a directory when only a single copy of the directory object is available; this approach has proved much more robust. Since the interface to the catalog manager was not affected by this change to the implementation of the catalog operations, no other system or application components had to be modified.

As another example, at the beginning of the C^2 Internet effort, it was not possible to use the multi-tasking facilities of the automatically generated manager software simultaneously with the Cronus large message facility. This defect was acceptable in the early versions of the manager development tools because the initial operations required only the small message capabilities; supporting task switches between fragments of a large message would have complicated implementation during the early experimental stages of development. This was changed also, to allow C^2 Internet managers to send back substantial amounts of data in response to operation invocations, while still maintaining the ability to respond to new operation invocations. The large message support capabilities were used in the development of the Cartographic data manager, which often must transmit large amounts of map data to its clients in response to a single retrieval request.

3.3. Application Development Tools

The implementation of a set of tools for object manager development is perhaps the single most important productivity enhancement mechanism provided for distributed Cronus applications. The C^2 Internet experiment was built exclusively using these tools and was the first major application to exercise them. It is clear that the number of managers comprising the C^2 Internet experiment could not have been successfully hand-coded given the limited resources available—the availability of the manager development tools made this possible.

The manager tools provide a number of functions:

1. **Create** and register a new type and its definition including new canonical types, operations and their parameters, and access control restrictions.
2. **Generate code** for the manager, including operation dispatching routines, message parsing functions, and canonical-to-internal format conversion routines (and vice versa).
3. **Generate procedure-call client interfaces** to invoke operations on objects handled by the manager.
4. **Generate documentation** in UNIX manual page format using the comments provided within the definition of the manager.
5. **Generate a type list** of currently defined Cronus type numbers and names.
6. **Generate an error list** of currently defined Cronus error code numbers and their corresponding printable strings.

Based upon the high-level specification the developer registers with the type definition service, the service will generate source code for invoking operations from within a client and for dispatching operations within a manager. Library functions support generic object operations such as creating and removing objects, and support storing the instance variables of the objects for which each manager is responsible. These tools also provide support for access control and replication through a combination of code generated from the developer's specification and functions provided by the library.

The support for access control and replication provided by the Cronus tools was especially effective. Access control checking for most operations is quite routine, requiring the retrieval of client privileges and the access control list for the intended object and then verifying that the client has sufficient rights to perform the operation. The tools allow the developer to customize and symbolically specify the rights which are required for a user to perform each operation and then provides code as part of the manager dispatching routines to perform the access checks.

Replication was also useful. The replication facilities offered by the tools operate by maintaining copies of replicated objects at all sites supporting managers for the given service. No special support is provided by the developer; all the coordination and control is performed by software provided by the tools. This was invaluable for expediting development, and shows that high-level specification of replication is a feasible and useful capability. However, replication of all objects at all sites does not scale well. In larger configurations a more controlled replication strategy, such as what the catalog manager supports, would be more effective. Also, the built-in strategy uses a simple timestamp to detect discrepancies among the copies; in the future, other detection mechanisms such as version vectors and voting schemes should be provided as specifiable alternatives to the current scheme.

The major drawback of these tools was their immaturity during the early stage of our development. The tools were initially built to facilitate the development of small set of system managers, and focused on the needs of those managers. Additional features were added to the tools as we began to use them for the C^2 Internet Experiment. Because the tools were so new, they were not as reliable or robust as we would have desired, and conventions that tools users were required to follow when writing software to combine with tool generated code often changed. Documentation was usually sparse and out-of-date, with many new features missing completely. And the diagnostic messages offered by the compiler were often limited and difficult to interpret. Application developers were able to overcome these shortcomings only because of their close working relationship with the system developers.

However, as a result of this effort the tools have evolved into a much more suitable base for application component development. We are largely satisfied with the functions they now provide for developing applications such as C^2 Internet. We do believe additional effort is needed to improve the reliability of the tools, particularly in areas such as error reporting. We also believe additional functions will need to be added to support larger applications and larger development teams. The need for the system, rather than developers, to manage sharing information such as canonical types descriptions and operation specifications will increase, as will the need to manage configurations in which some components have fallen out-of-date. These areas are now handled by limited mechanisms for sharing provided by the tools and by cooperation among the developers. However, additional changes should not introduce the substantial recoding that earlier changes required. Most of the extension of these tools can be done transparently for existing application and system managers built using the manager development tools.

3.4. User Interface

Cronus provides rather limited user interface utilities so the programming and command languages of the component hosts must often be used to build specialized user interfaces. This has the advantage that developers are encouraged to use the facilities available on a particular access host when building an interface or command component. However, this also means Cronus does not yet offer a uniform interface standard, especially one that is well integrated with the object oriented development methodology.

The C^2 Internet application was one of the first Cronus components to make extensive use of graphics in its user interface[4]. Because we expect that many other interfaces to systems such as C^2 will rely on graphics interfaces, it is worth considering user interface development support, integrated in a useful way with the Cronus object model and with the Cronus software development tools. In the case of C^2 Internet, the SunWindows graphics interface supported on the Sun workstation was used quite successfully, providing a display in which target reports and other data could be easily displayed against a cartographic background. However, this part of the application is not easily transported to other systems.

The flexibility afforded the developer by allowing the use of any available programming language also allows each developer to adopt personal, idiomatic user command formats. This was true in many cases for system command names: sometimes the command name was the name of the operation, e.g. *list*; and sometimes the command name was a mixture of words and abbreviations, e.g. *setwdir* for *set* *w*orking *dir*ectory. With regard to parameters, parameter formats often varied from one command to another. In particular, some commands required that the user indicate the type of any object whose name was given, other commands required that the argument be of a particular type. In developing the C^2 Internet Experiment we tried to avoid such inconsistencies and made efforts to build interactive menu and prompt driven programs for the principal user interfaces.

3.5. Development Testbed

The Cronus cluster located at BBN provided testbed support for all development activities performed under this effort. This testbed was comprised of Cronus support operating on several host systems of varying type and architecture. This Cronus cluster was under development and being maintained by the primary Cronus development project. The C^2 Internet Experiment used this environment concurrently with the primary project, both for Cronus support and to satisfy general application development and processing requirements.

One of the major hindrances to application development was the instability of the underlying Cronus system software on which the application was being built. This was caused by the inability to easily isolate application developers from Cronus DOS developers in the resource-limited environment. Typically, application and system development were conducted on the same set of host computers. Intermittent problems caused by ongoing changes to the system software often caused stable C^2 Internet components to fail. Such problems were difficult to diagnose largely because known changes to the system were often not recognized as relevant. This situation is more manageable now, as we begin to separate the system and application development environments and acquire additional hardware to enforce configuration management disciplines.

[4] The Cronus Monitoring and Control System also makes extensive use of graphics.

The obvious conclusion we have reached is that in an environment where multiple people will be developing both application and system software, it is very important to have a relatively stable, bug free, and complete version of the Cronus DOS and its associated support utilities and documentation. It is rarely the case, though, that sufficient resources are available for these purposes, especially for as yet unproven systems. Initially developing the application in the same environment as the system did lead to much more effective feedback to system developers regarding the needs of the application developers, and faster turn-around when addressing problem areas. In the future we plan to divide the host resources between system and application development and move toward a more formal release structure with formal bug reporting and repairing mechanisms in place.

3.6. System Configuration Maintenance

Configuration management of the services provided by the Cronus DOS was relatively simpler than that of components for the C^2 Internet Experiment project. The interfaces to most system services were well defined by the time client programs were developed to use them. Few system managers require interaction with managers for other system services to complete client requests. And, once a manager was installed on a host it was likely to remain there and be operational for a long period of time.

The management of the C^2 Internet Experiment components is somewhat more complex. When a client invokes an operation on an object, the manager for that object often has to invoke operations on other secondary objects before the initial request can be completed. The interfaces to C^2 Internet Experiment components evolved as the application was developed, sometimes interfaces where revised and frequently new functions were added to the interfaces. Also, because the collection of services was growing regularly, the placement of the service managers changed several times during the development process; old versions of managers might remain at inactive sites and need to be updated whenever the active managers where revised.

As a result of this increased complexity, the C^2 Internet experiment raised a number of interesting questions related to system configuration maintenance that had not been previously examined within the Cronus environment with smaller-scale client-manager and manager-manager interactions. The first of these was how to ensure that managers and clients were using the same specification for each operation supported by the manager. During early development operation specifications sometimes change; this happens less frequently once the initial version of the manager has been developed, but does happen occasionally. For a manager to successfully interpret an operation request, both the client and manager must be using the same description of the operations. To some extent this problem is remedied by automatically generated interface routines, which ensure that the operation message encodings match if both the client invocation and manager dispatch code were generated from the same description. For a client to talk to a manager, it merely uses the automatically generated interface. However the management of these automatically generated routines was not so straightforward. If a client wants to talk to a manager of a given type, and the interface for that type has changed, the client must know about the change.

The problems of occasional inconsistency between client and manager that we encountered during the course of C^2 Internet implementation required two complementary solutions to this problem. First, all interface routines for the managers relevant to the particular overall application are kept in a common up-to-date library. If a manager is updated, its interface routines in the library are also adapted. Thus when a client is updated, it will obtain (at compile time) the up-to-date interface routines from the library. The second addition detects operations invoked by out-of-date interface routines to guard against the manager being changed without the library being updated. When the manager and client routines are generated, a software *checksum* is included with the automatically generated code. This checksum is then compiled into the corresponding manager and interface routines. At run-time, the client includes this checksum value with each operation invocation message and the checksum is verified by the manager. If the manager discovers an incorrect checksum, it assumes the client is using an out-of-date specification for the operation and the operation is rejected.

Another problem in organization that was encountered in C^2 Internet was the sharing of data representations through Cronus canonical data types. For example, many C^2 Internet Experiment managers needed to have a common representation for the location of a simulated physical object: [latitude, longitude, and altitude]. We chose to represent locations and each of the necessary shared data structures as Cronus canonical data types. To ensure that all application components used the same representation for each data type, we defined a generic parent type associated with the overall application, in which all the shared representations where described. Most of the C^2 Internet Experiment object types where defined as children of this generic application data type, and inherited identical descriptions for the shared data types. This is an adequate solution for our present problems, but does not scale well as more and more shared data types are required; many object types, and thus managers for objects of those types, will inherit canonical types that they never use. A better solution that has been proposed is to allow *reference* canonical types, in which a particular manager may use a canonical type defined by another manager by explicitly importing the definition of that type.

3.7. Monitoring and Control

While the services provided by the Monitoring and Control System, such as event reporting and configuration management, were effectively used in the C^2 Internet Experiment application, the console interface was not at all effective for viewing and controlling the status of the application. The existing MCS console interface serves to illustrate the role of monitoring and control, and its possible effectiveness, but the responsiveness and stability of the component is not adequate for routine demonstrations or operational use. Its implementation was experimental in both the icon oriented interface it provided and in the underlying programming support that was used to implement it. Significant portions of the MCS are undergoing redesign and will need to be reimplemented to make it operational.

The data collection and evaluation tools of the MCS should also be extended. The focus of the current MCS was largely instantaneous status monitoring and control, rather than long term data collection and analysis. The availability of chart graphics for comparison and analysis of long term system performance would greatly improve the usefulness of the MCS as a system management tool.

3.8. Debugging Cronus Applications

One of the most difficult parts of developing a distributed application is its debugging. Not only does one have the difficulties associated with debugging a more traditional program, but additional complexity is added by having a number of components acting in harmony to perform some overall function. This section attempts to discuss some of the issues that were raised during the distributed application development and which will influence the development of a debugging tool for Cronus.

A variety of debugging facilities are used within the current Cronus application development environment:

- Existing **local debuggers** for the environments in which Cronus is built (adb, dbx, xmd, etc.).
- General purpose **operation-oriented interfaces** (i.e., the auth/ui family of programs[5]) for invoking a single operation at a time.
- **Object dumps** generated by invoking a particular operation on an object.
- **Logging** both by managers and by the Cronus kernel.
- **Hand-crafted test programs** oriented toward particular managers or applications.
- **Miscellaneous tools** that can be used with programs to assist in finding bugs (e.g., specialized versions of the malloc and free memory allocation routines).

These tools are adequate much of the time, when debugging a single manager or a single interacting client and manager. One thing that we are discovering in C^2 Internet, however, is that as we get more complex multi-manager interactions, it is more difficult to determine which component is at fault when something doesn't work as expected. Because of the size and complexity of distributed systems, application developers are often unaware of the underlying interactions necessary to perform the requests submitted by their applications. As a result, diagnosing intermittent problems often requires examining detailed logs kept by the Cronus kernel and system services.

Some system designers (e.g., the authors of "A Debugger for Message-based Processes" [Smith85], and "A Distributed Programs Monitor for Berkeley UNIX" [Miller86] take the view that a valuable tool for multi-process debugging is one that allows the programmer to observe and control the interaction between processes. This kind of debugger is oriented more toward finding bugs that occur when components interact, rather than finding bugs that are isolated in a particular component.

[5] See Cronus User's Manual [Sands86] for details of these programs.

In the Cronus environment, a tool would be desirable that could:

- Trace objects (i.e., signal a user when an operation is invoked on a particular object).
- Breakpoint multi-operation interactions on an operation-by-operation basis.
- Trace and intercept messages to or from a given host or manager.
- Trace and intercept messages with particular contents.
- Allow the formatted display of message contents.
- Allow the alteration of message contents.

Some problems arise with any approach to a debugging facility. The first of these is performance. Obviously interposing any kind of filter into the message path will increase the message transit time. However this should not be a major difficulty during debugging. What is more important is that when manager's aren't being debugged, the amount of additional overhead imposed by the presence of available but unused debugging mechanisms be minimal.

A second difficulty is the current use of timeouts to detect component failures. It would be helpful to have some kind of timeout cooperation in the debugging process. This is particularly important when a user wishes to 'breakpoint' a multi-operation transaction somewhere and then 'continue' it. If timeouts are not somehow controlled within the debugging environment, then the delay caused by a user examining the system's state at a breakpoint may cause the overall transaction to timeout and hence be unsuccessful[6].

Another problem is security. If the debugger has the ability to alter an operation (either the source address to make it appear to be from its actual sender, or the contents, in order to alter the receivers behavior), there is a chance that the facility will be abused.

Our experience shows that enchanced application debugging facilities are needed for system and application development. Because of the complexity of the problem, we believe additional experience with application development is required before significant development can proceed. As with the language development tools, we expect that debugging tools will be initially developed as a prototype and evolve as they are exercised by application developers.

[6] We have successfully experimented with such mechanisms in previous projects, such as the National Software Works, but have not yet added such mechanisms to Cronus.

3.9. Performance Improvements

In general, the performance of the underlying system in quantitative terms was adequate for our needs. This allowed us to focus on qualitative evaluation, as we had hoped to do. We did, however, develop several performance measurement tools to help pin-point delays in operation processing. These tools measured message transit times and profiled operation processing within managers and clients. These routines were used largely as a diagnostic tool to identify areas where the implementation of particular components should be improved and where operation protocols should be revised. The tools also indicated that an inappropriate amount of time was being spent encoding and decoding the data in messages; the libraries and application development tools were modified to improve the performance of these message translation activities.

3.10. Documentation

Most users are introduced to a computer system initially through its documentation. For various reasons, the Cronus documentation available to application developers during the initial implementation phase of the C^2 Internet Experiment was difficult to use.

The first and foremost of the difficulties was that there was no tutorial documentation for Cronus. Such a document might include introductory discussion of what Cronus is about, how the Cronus system software is structured, and how to navigate through the rest of the documentation. The suitability of the existing documentation at the time to application development was limited. Most of the documentation was written for system, rather than application, developers and was difficult for use by programmers whose primary purpose was to build a distributed application.

A second difficulty was that, although the system-level documentation was relatively complete, it was often out-of-date. To a great extent this was caused by the fast pace at which the system was developed: the documentation often lagged behind the actual state of the system. This is another consequence of the limited resources available in the development environment. A sound way to alleviate this problem in the future would be to enforce a stricter separation between the application and system development environments, maintain an up-to-date set of documentation for the former, and limit the rate of change in the application development environment. This could only be accomplished with a high degree of confidence now that we have completed the C^2 Internet demonstration application.

Finally, confusion was often caused by the format of the documentation itself. The initial Cronus standard of UNIX-style manual pages was adopted for convenience to the system developers producing and maintaining manual entries; it is not particularly easy for an application developer to use. Finding a subroutine that performs a particular function, even when one knows it exists, is quite difficult unless the name of the subroutine or the name of the subroutine package that includes it is known[7]. The Cronus software is organized in a layered architecture, both logically and in its implementation. A documentation style and organization that mirrored this layering would make considerable headway in simplifying the use of Cronus documentation.

[7] This is more a problem that the Cronus project inherited from UNIX by adopting the UNIX tools for manual maintenance rather than a problem inherent in Cronus itself.

4. CONCLUSIONS

4.1. Review

Our experience with Cronus in the course of the C^2 Internet Experiment has been very positive. From our experience, we believe Cronus provides a good unifying element for building distributed applications. Our initial goals have all been met:

1. We demonstrated that a prototype demonstration complex distributed application could be built naturally and with relative ease using Cronus and its support services.
2. The application seems relevant to the C^2 domain and makes considerable strides toward architecturally organizing these activities.
3. The demonstration uses a wide variety of Cronus mechanisms to support distributed system benefits. In particular, approaches to access control, survivability and resource management are successfully operational in the demonstration context.
4. The development served to build a feedback cycle from system users to guide system developers.

With a limited amount of effort, we were able to successfully implement a relatively complex distributed application simulation. Despite the newness of the Cronus system and the lack of appropriate support or documentation, the application development was successfully carried out by a mixture of novice and experienced programmers from both system and application programming backgrounds. Also, the initial architecture of the application required very little modification during the course of development. Collectively we believe that this provides validation that the Cronus object model and associated system, software development tools, and utilities are an effective approach to distributed application development.

The C^2 Internet Experiment application selected appears to be relevant to some initial intended uses of Cronus and appears to be relevant to future DoD goals. This was our belief after initial reviews of our proposed application architecture and demonstrations scenarios and has been confirmed by evaluations made by audiences who have seen the application demonstrated. We also believe the application can be easily extended, both by the addition of new components and by the enhancement or replacement of existing components. As such, the existing application provides a foundation for continuing application evolution and for the integration, experimentation and evaluation of new and enhanced C^2 and technology components.

The application itself uses a variety of Cronus mechanisms to support distributed system benefits such as survivability, scalability, and resource sharing and management. In many cases, facilities offered by the application development tools were used directly. This was particularly true of the replication and access control facilities. In other cases, more primitive mechanisms were used to support more specific needs of an application component, prior to their inclusion in the application tools set. The underlying Cronus system services were also used extensively. The catalog is one of the most visible of these components, since it provides symbolic names for sensors and mission data reports which would otherwise be referenced by a numerically encoded unique identifier, or an ad hoc application specific mechanism.

The feedback offered by the C^2 Internet Experiment to Cronus developers was instrumental in guiding current and future system development. In particular, additional emphasis was placed on tool development since the underlying IPC mechanisms and system services seemed adequate for our use. As a result, the development tools became much more useful and substantially improved our productivity in developing the application. Many other bugs and weaknesses in the underlying system were corrected. Additional areas, such as documentation and debugging, and the extension of Cronus to other computer systems and support technologies were identified for future work.

4.2. Future Directions

Several areas were identified as suggestions for future work. Development in these areas will make the services and tools available to application builders more powerful and accessible. Some of these areas are currently being addressed or are planned for work under the current RADC program.

1. Documentation is needed to support new users. A tutorial introduction to Cronus is planned. We should also consider developing appropriate training materials and a training program for new users, as well as better documentation at all levels.
2. The current demonstration is being expanded to a multi-cluster demonstration scenario. Although demonstration scenarios still involve only resources from a single cluster because no other clusters were available during the course of early development, the scenarios will be extended to employ resource sharing among multiple clusters[8].
3. Addition of new resources to the Cronus clusters such as LISP machines, multiprocessors, and personal computers to enhance the opportunity for functional specialization. A pilot investigation of integrating a Symbolics LISP machine has already begun. This includes adapting the application development tools to support both client operation invocation and manager operation support within the LISP language. C^2 Internet Experiment components that exploit the advantages of such systems should be designed and integrated into the demonstration application.
4. Separate facilities for application development and Cronus system development should be created. Some effort is being expended in this area, but limited hardware resources make it difficult. We are also working to improve the reliability and stability of the Cronus DOS and tools. As part of this effort, we are also moving toward a more formal release structure for software distribution to sponsor sites.

[8] A cluster installed at RADC in Rome, NY, currently communicates through the DARPA Internet and the Wideband Satellite Network to the original Cronus cluster located at BBN in Cambridge, MA. We have presented initial demonstrations of remote access to BBN resources from the RADC cluster, including access to symbolic processing and database services. We are working to improve the reliability of the intercluster interactions so that they can be routinely used in demonstrations and so the more extensive resource sharing can occur between the two clusters.

5. The current MCS system serves to illustrate the use of distributed monitoring and control of the Cronus DOS and applications, but is not functional enough, nor is its performance adequate, to be used in operation or flexibly in new demonstration environments. Monitoring and Control System support should be made more reliable and should be expanded to serve in a more functional role. It should also be expanded to provide better support for performance monitoring and control of resource management.

6. Debugging support for application software development in distributed heterogeneous computer environment should be improved, as well as additional tool support for other parts of the software development cycle, especially to support larger, multi-person projects.

Larger research areas are also indicated. The availability of an integrated database management would simplify many component functions which are not easily rendered in object terms. This includes the Mission Data Management System, where the object model was effectively applied to support a database storage and retrieval function, but which could have been more readily and flexibly implemented using a database system. Such a system would also have afforded a much richer set of query requests at little implementation cost, essentially requiring the interface manager to compose the appropriate query syntax rather than supporting all the record scanning and selection. We anticipate development work will begin in this area under a separate contract quite soon.

As the system grows to span several clusters, integrated distributed system security also becomes an issue. The existing system supports discretionary controls, although a system expert can undoubtedly circumvent this protection. The system does not support administrative distinctions between clusters, so members of one cluster can freely make use of resources within another cluster, subject to the existing access control mechanisms. Beyond this is the issue of mandatory, multi-level security mechanisms, which are not at all supported within the existing system. A project to explore and devise strategies for ensuring mandatory, multi-level security within the object/operation invocation framework of a distributed operating system is currently underway. Issues of this level of control have not been relevant to the current effort.

We are encouraged by the success we have had with the Cronus testbed, both with its support for high-level application development and with the ease with which we have been able to modify and improve the underlying system support base. We believe that Cronus will provide a sound and effective foundation for distributed application development in the future. We anticipate that additional testbed applications will be developed, that new technological components with be incorporated, and that the number of participants will grow in the near future. This practical, ongoing testbed validation of Cronus capabilities, combined with continued research into new areas, form a strong program for improving the support offered to distributed applications developers working within the Cronus intercluster environment.

5. REFERENCES

[Berets85] James C. Berets, Ronald A. Mucci, and Richard E. Schantz, *Cronus: A Testbed for Developing Distributed Systems,* 1985 IEEE Military Communications Conference (October 1985).

[Berets86] James C. Berets, Ronald A. Mucci, Richard E. Schantz, and Kenneth J. Schroder, *C2 System Internet Experiment: Functional Definition,* BBN Laboratories, Technical Report No. 5942 (May 1985, Revised May 1986).

[Berets86a] James C. Berets, Ronald A. Mucci, and Kenneth J. Schroder, *C2 System Internet Experiment: System/Subsystem Specification,* BBN Laboratories, Technical Report No. 6248 (July 1986).

[Berets86b] James C. Berets, Ronald A. Mucci, Richard E. Schantz, and Kenneth J. Schroder, *C2 System Internet Experiment: User's Manual,* BBN Laboratories, Technical Report No. 6249 (May 1986).

[Gurwitz86] R. Gurwitz, M. Dean, and R. Schantz, *Programming Support in the Cronus Distributed Operating System,* Sixth International Conference on Distributed Computing Systems (May 1986).

[Miller86] Barton P. Miller, Cathryn Macrander, and Stuart Sechrest, "A Distributed Programs Monitor for Berkeley UNIX," *Software - Practice and Experience* 16(2) pp. 183-200 (February, 1986). Presented at the 5th International Conference on Distributed Computing Systems

[Sands86] R. Sands and K. Schroder, eds., *Cronus User's Manual,* BBN Laboratories, Technical Report 6180 (February 1986).

[Schantz85] Richard E. Schantz and Robert H. Thomas, *Cronus, A Distributed Operating System: Functional Definition and System Concept,* BBN Laboratories, Technical Report 5879 (June 1982, Revised January 1985).

[Schantz86] R. Schantz, R. Thomas, and G. Bono, *The Architecture of the Cronus Distributed Operating System,* Sixth International Conference on Distributed Computing Systems (May 1986).

[Smith85] Edward T. Smith, "A Debugger for Message-based Processes," *Software - Practice and Experience* **15**(11) pp. 1073-1086 (November, 1985).